Chapter 8: Fundamental Audio Processing

In earlier chapters we describe how sound is produced, what the basic physical properties of sound are, and how one can quantify them. In this chapter we will discuss some basic operations originating from signal processing that are common initial steps of many algorithms -- whether they be audio enhancement, audio compression, or acoustic content analysis. In fact, they are so common that they are often not even mentioned in research papers when discussing novel algorithms. We will later describe how one can use this knowledge to compress speech and sound and how to use they build the basis for content analysis. Additionally they exist in very similar shape in image and video processing as explained in later chapters.

Basic Frequency Domain Transformations

In this digitized age, all the signals we are dealing with as a computer scientist have been sampled over time. In other words, we receive a stream of numbers that are representative of the strength of the signal at certain time points. This representation is therefore also called time-domain, as the x-axis of a graph showing a signal this way would be time. In frequency domain, all the points of the x-axis represents a certain frequency. Visualized as a graph, the diagram would therefore show how the signal varies over frequency. Typically, just like in time domain, the y-axis would show the energy or amplitude of the signal.

A given signal can be converted between the time and frequency domains with a number of mathematical operations, here we shortly present one of them. Chapter XXX (spectral compression) presents another one. The explanation given in this chapter is by no way comprehensive and only meant to refresh your memory. If the concept of frequency space and the math underlying it is not familiar to you, we recommend consulting the literature referenced at the end of the chapter.

Discrete Fourier Transform (DFT)

The theory of the Fourier transform goes back to 1822, when Jean-Baptiste-Joseph Fourier found that any function can be described as a sum of sine and cosine functions. Periodic functions need a finite set of sine and cosine functions and aperiodic functions an infinite number of sine and cosine functions. The original Fourier transform assumes continuos functions rather than discrete sequences and an open interval domain. Therefore, in practice, the discretized version of the Fourier transform results in artifacts at the borders of the sequence, which have to be come over by windowing strategies. Both the Discrete Fourier Transform as well as the Discrete Cosine Transform are in theory reversible without signal loss. In practice though, numerical constraints make the transforms often slightly lossy. However, unless noted otherwise, we will assume the transforms to be lossless.

The input to the DFT is a sequence of N real or complex numbers x_0, ..., x_{N-1} which is transformed into a sequence of N complex numbers X_0,..., X_{N-1} by the DFT according to the following equation:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

where i is the imaginary unit and $e^{\frac{2\pi i}{N}}$ is a primitive N'th root of unity. The inverse transform is given by:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

The direct application of the equations above as implemented in the pseudo-code requires $O(N^2)$ operations. However, it is possible to execute the same computation with only $O(N \log N)$ complexity by factorizing the equation and reusing intermediate result by applying dynamic programming. The resulting algorithms are called Fast Fourier Transform and Fast Cosine Transforms. In practice, mostly these algorithms are implemented into actual codecs. In addition, the DFT and DCT (see below) computations are parallelizable which results in speed gains on manycore architectures.

The following pseudo-code can transform a given signal into frequency domain using the Discrete Fourier Transform. The algorithm is called the Cooley-Tukey Algorithm, it is the most common algorithm to calculate the Fast Fourier Transform.

```
// Input: A number N of samples X. N must be an integer power of two.
//        s the stride
// Output: The Discrete Fourier Transform of X in the array Y.
FFT(X, N, s)
    IF (N==1) THEN
          Y[0]:=X{0] // Trivial size-1 case: Sample = DC coefficient
    ELSE
          Y[0,...,N/2]:=FFT(X,N/2,2*s) // DFT of (X[0],X[2s},X[4s},...)
          Y[N/2,...,N-1]:=FFT(X[s,...,N-1],N/2,2*s)
                                  // DFT of (X[s],X[s+2s],X[s+4s],...)
    FOR k:=0 TO N/2 // Combine the two halves into one
          t:=Y[k]
          Y[k]:=t+exp(-2*pi*k/N)*Y[k=N/2]
          Y[k+N/2]:=t-exp(-2*pi*k/N)*Y[k=N/2]
FFT := Y
```

The elements of the converted signal are called coefficients. The first coefficient corresponds to the *sin(π/2)* which is 1. Therefore it is called DC coefficient (from the electrical engineering interpretation DC = direct current), the rest of the coefficients are called AC coefficients (again, AC = alternating current).

The Convolution Theorem

Mathematics defines convolution as an operation on two functions *f* and *g*, that produces a third function. This third function is typically viewed as a modified version of one of the original functions, e.g. an image, a video, a set of measurement samples, or an audio track modified by a second function, usually called filter. The principle of convolution is a very basic concept appearing in many research fields, including statistics, physics (especially in differential equations), electrical engineering, computer vision, and multimedia computing. The mathematics behind convolutions fills books, so again, the reader is suggested to dive deeper into the literature

for further information. In multimedia computing, the most important fact to know about convolutions is the convolution theorem. We already introduced the notion of frequency space in Chapter XXX (advanced compression). Here we introduce a fundamental correspondence between the time space and the frequency space: the Convolution theorem. It states that a convolution filter in one domain (e.g. the time space) equals point-wise multiplication in the other domain (e.g. frequency space).

Let *f* and *g* be two functions and *f* * *g* their convolution. Note that the * denotes convolution in this context, and not multiplication). Let $\mathcal{F}$ denote the Fourier, so $\mathcal{F}\{f\}$ and $\mathcal{F}\{g\}$ are the Fourier transforms of f and g, respectively. Then

$$\mathcal{F}\{f * g\} = \mathcal{F}\{f\} \cdot \mathcal{F}\{g\}$$

where the dot denotes point-wise multiplication. The other direction is also true:

$$\mathcal{F}\{f \cdot g\} = \mathcal{F}\{f\} * \mathcal{F}\{g\}$$

By applying the inverse Fourier transform, one can write:

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

Note that mathematically speaking the relationships above are only valid for the canonical form of the Fourier transform. In practice, however the transform may be normalized in other ways, in which case constant scaling factors may appear in the relationships above. Most importantly, the theorem allows us to design convolution filters in either time or frequency space whichever is more convenient for the concrete problem. In practice, this is exactly what is done and this is also why some of the frequently used filters presented in this chapter have their "home" in frequency space and some others seem to be mostly applied in time space.

Linear Filters

A very common type of filter useable in many situations and in many domains is the linear filter. The linear filter applies a linear operator (as defined in linear algebra) to a time-varying input signal. An operator is linear if the following conditions hold:

Let *V* and *W* be vector spaces over the same field *K*. A function $f: V \to W$ is said to be a *linear operator* if for any two vectors *x* and *y* in *V* and any scalar *a* in *K*, the following two conditions are satisfied

1) Additivity: $f(x+y) = f(x) + f(y)$

2) Homogenity (of degree 1): $f(ax) = af(x)$

It follows from the definitions that *f(0)=0*.

The above mentioned mathematical properties have several advantages in real-world applications. For example, because of the homogenity rule, it does not matter wether a linear filter is applied before or after amplification of a signal. The same applies for the additive mixture of two signals: The additivity makes sure that it does not matter whether the filter is first applied to the each of the mixture components before the mix or after the mix to the added components. This is especially useful to generalize a 1-dimensional linear filter to more than one dimension: One can just apply the filter to each dimension individually. Because of these properties, many real-world circuits and filter formulas aim to be linear. Distortions are often caused by deviations from the linearity assumption of a real-world equipment, such as amplifiers (which are essentially multipliers of the signal in time space).

Linear filters are further divided into two classes: infinite impulse response (IIR) and finite impulse response (FIR) filters.

FIR filters (which may only be implemented on a discrete time scale) can be described as a weighted sum of delayed inputs. If the input becomes zero at any time, then the output will eventually become zero as well, as soon as enough time has passed so that all the delayed inputs are zero, too. Therefore, the impulse response lasts only a *finite* time. In an IIR filter the set of time where the output is non-zero will be unbounded; the filter's energy will decay but will be ever present. Most of the filters were initially developed as analog electrical circuits. So the field of electrical engineering predominantly discusses the various aspects of filter design. The mathematical discipline behind it is called linear time-invariant (LIT) system theory. The following section discusses some commonly used filters.

Common Linear Filters

Linear filters are often used to eliminate unwanted frequencies from an input signal or to select or enhance a set of desired frequencies among many others. As the reader might already infer from past chapters, these operations are among the most important. Common types of linear filters include the low-pass filter, which passes frequencies below a threshold, a high pass filter passes frequencies above a threshold, and a band-pass filter passes frequencies of a certain band, i.e. between two frequency thresholds. Similarly, pre-emphasis amplifies a certain band (see Chapter XXX). A band-stop filter passes frequencies except a limited range, i.e. it is the reverse of the band-pass filter. A special, often used, band-stop filter is the so-called notch-filter. Notch-filters have a particularly narrow stop band and are often used in audio equipment, one example being the anti-humm filter that filters electro-wire-induced humming between 59 and 61 Hz or 49 and 51Hz (depending on the country's electrical specifications). The following example implements a typical band-pass filter:

```
// Input: 4096 audiosamples s[] sampled at 44100 Hz,
// lower boundary frequency lf, upper boundary frequency uf,
// an FFT function, an inverse FFT function (see Chapter XXX)
// Output: A filtered audio signal
bandpass(s[],lf, uf , fft, ifft)
    fs[][]:=FFT(s) // convert to frequency space with blocksize 4096,
                   // 2nd index 0-real value, 2nd index 1 = complex value
    FOR i:=0 TO 2048: // exploit symmetry
          IF ((44100/4096)*i<lf) OR ((44100/4096)*i>hf): // bandpass
```

```
                fs[i][0]:=0
                fs[i][1]:=0
                fs[4096-i-1][0]:=0
                fs[4096-i-1][1]:=0
    s:=iFFT(fs) // back to time space
bandpass := s[]
```

Equalizers use combinations of band-pass filters to extract individual bands that can then be amplified or suppressed according to user settings. Equalizers are part of virtually any home audio system and can be used to fine-tune the frequency spectrum of an audio signal, e.g. more bass can be added, and speech can be made more intelligible against background noise. The wrong settings, however, can degrade audio quality. Since the individual bands can be easily visualized using sliders, the filter is often called "graphic equalizer". Figure 1 shows a graphical equalizer that is implemented by simulating a 10-band octave filter array conforming to ISO Recommendation 266. The numbers on the bottom show the center filter frequency in each band and the sliders manipulate the energy in each band. The filters use a Gaussian curve which allows the equalizer to operate smoothly across the different bands. The center frequency occurs at the top of the Gaussian curve and is the frequency most affected by equalization. It is often notated as f_c and is measured in Hz. The 11th slider on the left regulates pre-amplification of the signal before it enters the filter bank.

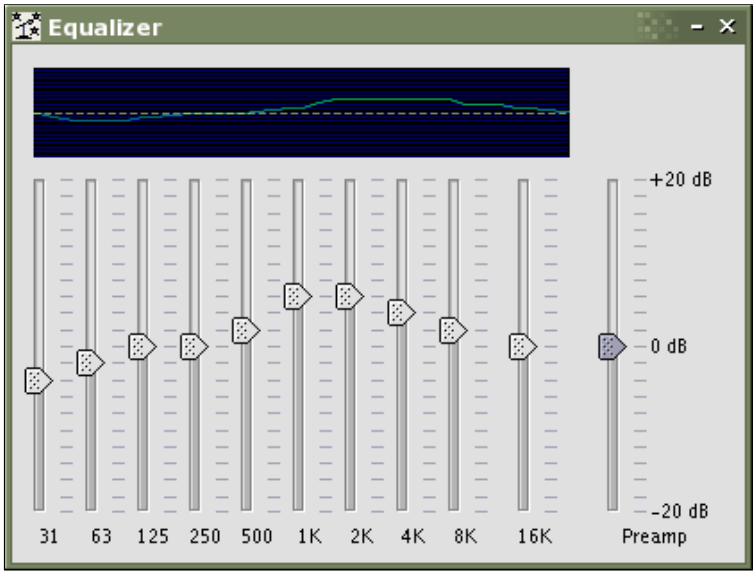

Figure 1. A graphic equalizer in a typical setting used for enhancing the intelligibility of speech.

Non-Linear Filters

Not all filters are linear. In fact, most filters are not -- but of course that does not mean that they are not useful. In the following we will present some important non-linear filters.

The first filter we discuss is so-called dynamic range compression (DRC). This filter often results in surprising perceptual enhancements of the audio signal. The analog process of DRC, sometimes simply called compression, is not to be confused with audio compression as discussed

in Chapter XXX, even though DRC can lead to better data compression rates. DRC is a process to is a process that reduces the dynamic range of an audio signal, i.e. it narrows the difference between high and low time levels. DRC is applied by running an audio signal through a dedicated electronic hardware unit or, nowadays, mostly through software. In the context of audio production, the hardware unit is often simply referred to as a "compressor". A DRC is a volume control filter. Two types of compressors exist: Downward compressors work by reducing loud sounds over a certain time threshold while lower amplitude sounds remain untreated. Upward compressors amplify sounds below a threshold while loud signals remain unchanged. In both cases, the dynamic range of the audio signal is reduced. DRCs are often used for aesthetic reasons or sometimes to deal with technical limitations of audio equipment, e.g. to avoid clipping.

Dynamic range compression often improves audibility of audio in noisy environments when background noise overpowers quiet sounds. In these cases, compressors are tuned so that they reduce the level of the loud sounds but not the quiet sounds the level can be raised to a point where quiet sounds are audible without loud sounds being too loud. An important thing to note is that analog compressors always decrease the signal to noise ratio. While this is sometimes acceptable for human perception it is often a problem for machine learning algorithms. Figure 2 shows the functionality of a typical compressor.

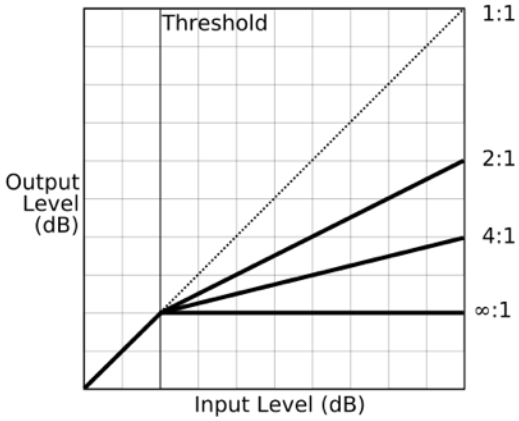

Figure 2. Relationship between input level, output level, and gain reduction in a set of example downward dynamic range compressors. This particular type of DRCs amplifies the signal below a threshold amplitude and then dampens the signal above the threshold amplitude.

Building compressors, especially as analog circuits, can be complicated as they require some manual fine tuning. For example, usually, the bend in the response curve around the threshold is often smoothed to be a rounded edge rather than a sharp angle as in Figure 2. Often it is desirable that a compressor provides a control over how quickly it acts as the volume pattern of the source material that is modified by the compressor may change the character of the signal quite noticeable. Some compressors therefore allow to define an attack and a release phase, similarly to the attack and release phases of a synthesizer (see Chapter XXX sound). Another difference is between peak sensing and RMS-energy sensing (compare Chapter XXX speech compression). A peak sensing compressor responds to the level of the input signal instantaneously while an RMS-energy compressors responds to the energy of the signal over a small time window. The second

type of compressor usually resembles perception of volume more closely. Other factors include the way a compressor reacts to stereo signals or if it reacts to different sub-bands differently (multi-band compressor). In the digital age, creating compressors doesn't involve fiddling around with complicated control circuits anymore, still tuning might be necessary. The design of a simple software compressor will be left as an exercise. Downward and upward DRCs might be combined using two or more different thresholds to form a variable-gain amplifier. Figure 3 shows a typical signal before and after the application of a variable-gain amplifier.

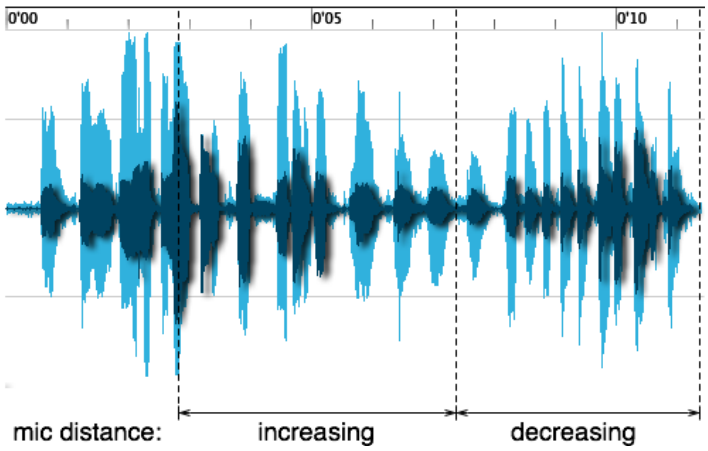

Figure 3. An example application of variable-gain amplifier. When the speaker turns away from the microphone, the audio gain goes down and with the mouth approaching the microphone the gain raises again (darker signal). Using DRC, the overall gain is higher and the microphone distance differences are leveled out more effectively (lighter signal).

Multiband variable-gain amplification is often designed to approximate the Mel or Bark curves (see Chapter XXX). This can be used to normalize sound files in a collection that have been recorded at different levels to about the same audible level. This function is often part of both software and hardware MP3 players.

Another set of very important non-linear filters are resampling (subsampling and supersampling) filters. Subsampling is the process of reducing the sampling rate of a signal. This is regularly done, for example, when a smaller thumbnail of an image is created or when compact disc audio at 44,100Hz is downsampled to 22,050Hz before broadcasting over FM radio. Since downsampling reduces the sampling rate, one must be careful to make sure the Nyquist sampling theorem criterion is maintained otherwise aliasing will occur (see Chapter XXX). If a signal has been recorded at a much higher sampling rate (e.g. for a music CD to be transmitted over FM radio) it is very likely that the original signal will contain frequencies higher than can be represented by the downsampled signal. To ensure that the sampling theorem is satisfied, a low-pass filter has to be used as an anti-aliasing filter to reduce the bandwidth of the signal before the signal is downsampled. When downsampling by integer factors (2,3,4, etc.) all one has to do is low-pass filter the signal and then pick every n-the sample (n being the downsampling factor) from the low-pass filtered samples. When downsampling to rational fraction m/n, the signal should be supersampled by factor m first, so that it can be downsampled by factor n in the second step.

Upsampling or supersampling is a bit more tricky. Supersampling is the process of increasing the sampling rate of a signal. For instance, upsampling raster images such as photographs means increasing the resolution of the image. Of course, objectively information cannot really be won by algorithmic upsampling since and the supersampled signal satisfies the Nyquist sampling theorem if the original signal does. So the basic supersampling algorithm is called zero-stuffing: Insert as many zeros as needed between two original samples and then apply low-pass filter with the threshold being the Nyquist limit frequency of the original signal. However, more elaborate algorithms try to model the underlying continuos signal from the original samples to then sample it again in a higher sampling rate, trying to convert alias into real information. These filters are often called interpolation filters.

The most basic interpolation filter is the constant interpolation filter: Repeat each value n times, with n being the upsampling factor and the apply a low-pass filter. Linear interpolation, is almost as basic: Instead of duplicating each sample, it assumes a linear function between each two original samples and creates as many samples as needed between two original samples as required by the upscaling factor. Generally, linear interpolation takes two data points, (x_a,y_a) and (x_b,y_b), and the interpolant is given by:

$$y = y_a + (y_b - y_a)\frac{(x - x_a)}{(x_b - x_a)}$$

between the point (x,y). Since we already know that sounds waves are in general of a sinus shape rather than linear, these two (too) simple technique will create (sometimes audible) artifacts. Therefore, other than linear functions are often used, for example, polynomials. Polynomial interpolation is a generalization of linear interpolation. Generally, if with n data points, there is exactly one polynomial of degree at most $n-1$ going through all the data points. However, calculating the interpolating polynomial is computationally expensive compared to linear interpolation. Suppose that the interpolation polynomial is in the form

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

We want $p(x)$ to interpolate all data points. In other words:

$$p(x_i) = y_i \qquad \text{for all } i \in \{0, 1, \ldots, n\}\,.$$

By substituting equation p(x) we get a system of linear equations in the coefficients a_k, which in matrix form reads:

$$\begin{bmatrix} x_0^n & x_0^{n-1} & x_0^{n-2} & \ldots & x_0 & 1 \\ x_1^n & x_1^{n-1} & x_1^{n-2} & \ldots & x_1 & 1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ x_n^n & x_n^{n-1} & x_n^{n-2} & \ldots & x_n & 1 \end{bmatrix} \begin{bmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}.$$

This equations can be solved by Gaussian elimination. Figure 4 compares the three approaches for a sample signal.

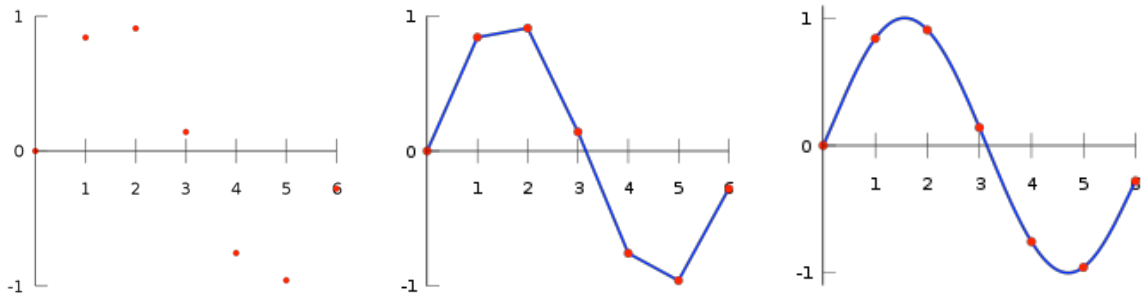

Figure 4. Left: original signal, middle: linear interpolation, right: polynomial interpolation. Note that even though the polynomial interpolation looks similar to what the original signal might have looked like, there is an infinite number of other possibilities how the original (continuos) signal might have looked like.

Because of the high complexity of polynomial interpolations, Spline interpolation is sometimes used, which implements piecewise polynomial interpolation with a low-degree polynomials. Spline interpolation is often used to interpolate vector graphics efficiently but not seen so often in audio processing. In order to avoid aliasing and further distorting the signal by up- or downsampling it, polyphase filters are often used in audio processing. Polyphase filtering divides the signal into different sub-bands which are treated independently. Using the critical subbands (see Chapter XXX) for example, can help reducing the effects of human audible aliases.

Filter by Example

Especially for noise reduction it is often desirable to be able to eliminate a very specific part of the signal that is not necessarily constant in frequency range or might not be sinusoid. For example, when digitizing a CD from an old vinyl record, scratching noise of the of the pickup might cover a range of frequencies. These frequencies might vary by phonograph, by disc, and maybe by piece of music. Also, the spectrum of the frequencies might be so large that using a band-stop filter to eliminate them all might distort the signal significantly and way beyond the noise that is to be eliminated. Therefore, a common technique to eliminate such noise is to perform the digitization normally, create a noise fingerprint based on a region of otherwise silent parts of the source, and then apply a technique call spectral subtraction. Spectral subtraction works under the assumption that noise has been added to the signal and therefore literally subtracts the noise fingerprint from the actual signal in frequency space. In other words, both the noise fingerprint and the signal are converted to frequency space and then the energy values of the noise fingerprint are subtracted from the signal. Often the noise fingerprint is pre-amplified or reduced depending on the situation. The signal might then be converted back to time space. The popularity of spectral subtraction is largely due to its relative simplicity, ease of implementation, and effectiveness especially in cases where complicated but constant noise-patterns make filtering with band-stop filters cumbersome. Figure 5 shows the effect of spectral subtraction. The downside of spectral subtraction is, of course, that it can seriously distort the signal and is not applicable in all cases.

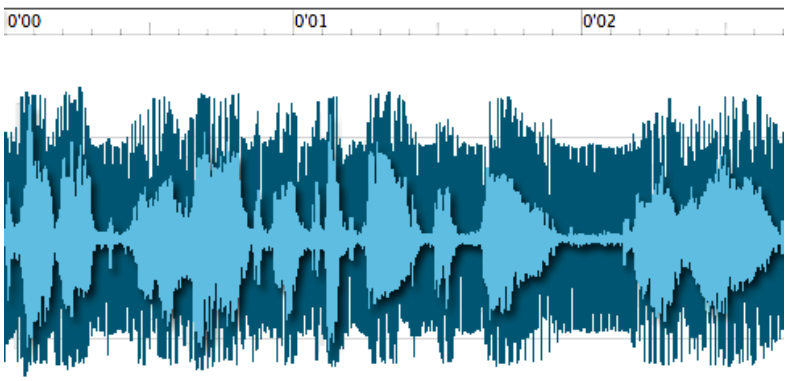

Figure 5. Three seconds of a speech signal (time space) with a 100Hz-sine-like humming before (dark) and after spectral subtraction (light). Humming is a frequent audio distortion when recording in situations that require long wires.

Another, very commonly used approach to reducing noise based on an example signal has been proposed by Norbert Wiener in 1949 and is consequently called Wiener Filter. Its purpose is to reduce the amount of noise present in a signal by comparison with an estimation of the desired noiseless signal. The filters presented before are designed to produce a desired frequency or amplitude response. However, the Wiener filter takes a different approach: One is assumed to have knowledge of the spectral properties of the original signal and the noise. Wiener filters are characterized by the assumption that the signal and (additive) noise are a stationary linear stochastic process with known spectral characteristics and known autocorrelation and crosscorrelation. Now the Wiener Filter seeks the linear filter whose output would come as close to the original signal as possible using the least mean-square error as optimization criterion. The input to the Wiener filter is assumed to be a signal *s(t)* corrupted by additive noise n(t). The output *s^(t)* is calculated by means of a filter *g(t)* using the following convolution:

$$\hat{s}(t) = g(t) * [s(t) + n(t)]$$

The error is consequently defined as:

$$e(t)=s(t)-s^{\wedge}(t)$$

which can be formulated as the quadratic error:

$$e^2(t)=s^2(t)-2s(t)*s^{\wedge}(t)+s^{\wedge 2}(t)$$

By applying the convolution theorem and other math, one can formulate the underlying optimization problem in frequency space as follows:

$$G(s(t))=|X(s(t))|^2/((|X(s(t))|)^2+(|N(s(t))|)^2)$$

Where *X(s(t))* is the power spectrum of the signal s(t) and *N(s(t))* is the power spectrum of the noise and *G(s(t))* is the frequency-space Wiener filter. If the shape of *N(s(t))* is known, an arbitrary optimization algorithm may be used. In practice the main problem is that accurate estimates of *N(s(t))* do not exist. Therefore, often *N(s(t))* is simply assumed constant at the level

of the signal to noise ratio. The following pseudo-code implements the calculation of a simple Wiener Filter this way:

```
// Input: An signal S[], an SNR snr, and a fingerprint P[]
// Needs: FFT function fft(), inverse FFT function ifft()
// Output: Wiener Filter F[]
Wienerfilter(S,snr)
    const:=1/(snr*snr)
    fft_F:=[][] // real part in fft_F[][0], imaginary in fft_F[][1]
    fft_S[][]=fft(S) // real part in fft_S[][0], imaginary in fft_S[][1]
    fft_P[][]=fft(P) // real part in fft_P[][0], imaginary in fft_P[][1]
    FOR i:=0 to length(fft_P):
          denominator:=fft_P[i][0]*fft_P[i][0]+fft_P[i][1]*fft_P[i][1]+const
          fft_F[i][0]:=fft_P[i][0]*fft_S[i][0]+fft_P[i][1]*fft_S[i][1]
          fft_F[i][1]:=fft_P[i][0]*fft_S[i][1]-fft_P[i][1]*fft_S[i][0]
          fft_F[i][0]:=fft_F[i][0]/denominator
          fft_F[i][1]:=fft_F[i][1]/denominator
    F:=ifft(fft_F)
    WienerFilter:=F
```

Graphical Filters

Some filters can be useful in practice even without changing the input signal. For example, when they help humans to adjust the input signal. One of the most frequently used examples of such a filter is the so-called VU meter. A VU meter is often included in audio equipment to display the signal level in so-called Volume Units (which are proportional to the energy of the signal). VU meters are built to intentionally measure the volume slowly by averaging out peaks and troughs of short duration to try to reflect the perceived loudness of the material. VU meters are standardized by IEC 268-10:1974. Figure 2 shows a typical VU meter that displays both the peak signal and the average signal level. It also counts clippings (a clipping is defined to occur when the signal reaches more than 98% of the maximum allowed range). The average gain level is measured by calculating the root-mean-square value of a time window of 250ms. The value ages with the last three measurements. The ideal recording maximizes the average signal without causing overrun. Original VU meters were analog tools.

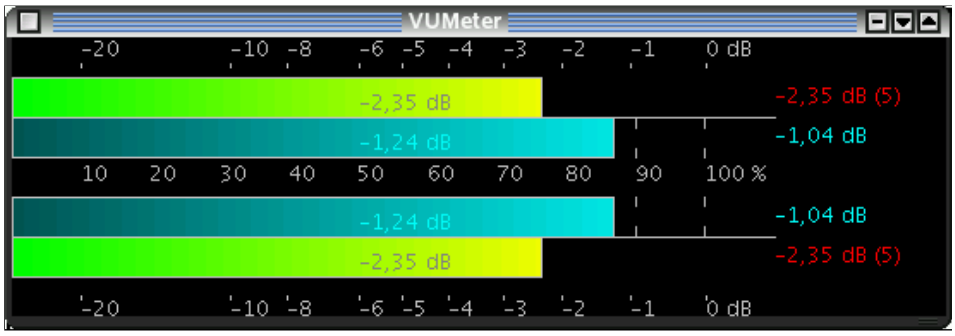

Figure 2: A screenshot of a typical (software) VU meter. This view shows a VU meter in stereo mode with a mono signal fed in. The inner bars show the average gain while the outer bars show the peak gain. Clippings are counted and displayed in red next to the peak gain meter.

Literature

Porat, Boaz (1997). *A Course in Digital Signal Processing*. New York: John Wiley
Brown, Robert Grover; Hwang, Patrick Y.C. (1996). *Introduction to Random Signals and Applied Kalman Filtering* (3 ed.). New York: John Wiley & Sons
Oppenheim, Alan V.; Schafer, Ronald W.; Buck, John A. (1999). *Discrete-time signal processing*. Upper Saddle River, N.J.: Prentice Hall. pp. 468–471.
Rabiner, Lawrence R., and Gold, Bernard, 1975: Theory and Application of Digital Signal Processing (Englewood Cliffs, New Jersey: Prentice-Hall, Inc.)
Williams, Arthur B & Taylor, Fred J (1995). *Electronic Filter Design Handbook*. McGraw-Hill.

Web Links

Fast Fourier Transforms, online book edited by C. Sidney Burrus, with chapters by C. Sidney Burrus, Ivan Selesnick, Markus Pueschel, Matteo Frigo, and Steven G. Johnson (2008):
http://cnx.org/content/col10550/latest/
FIR Filter FAQ:
http://www.dspguru.com/dsp/faqs/fir

Standards

IEC 268-10 1974 (Peak Programm Meter, Type 1)
ISO Recommendation R. *266*-1975: Preferred frequencies for acoustical measurements

Research Articles

Fourier, Joseph. (1878). *The Analytical Theory of Heat*. Cambridge University Press (reissued by Cambridge University Press, 2009; ISBN 978-1-108-00178-6)
Wiener, Norbert (1949). *Extrapolation, Interpolation, and Smoothing of Stationary Time Series*. New York: Wiley

Exercises

1. Implement a simple program that demonstrates the convolution theorem.
2. Smoothing a signal is a low-pass filter. Explain why.
3. Invent some simple mathematical functions for dynamic range compression.
4. Implement a graphic equalizer as described in the chapter. Discuss how wrong application of the tool can distort the audio based on experimentation with your implementation.
5. Imagine applying a graphic equalizer to an image. Explain what would change in the image in the image when you turn the sliders of the lower, mid, and higher frequencies up and down.
6. Older soundcards in the lower price segment often used simple averaging of the sample values for downsampling. Explain what the problem with this approach is.
7. Explain using examples why information cannot be won back by supersampling.
8. How can spectral subtraction work in time space?
9. Implement a Wiener Filter based on the pseudo code presented here. How is the filter applied and what can be improved?
10. Think of a use case for a VU meter and explain how you would want it to work for this use case.